The DARPA Advanced Logistics Project

Todd Carrico	Steve Milligan	Leo Pigaty	V.S.Subrahmanian
DARPA	BBN Technologies	Los Alamos Tech. Assoc.	University of Maryland
tcarrico@darpa.mil			

Abstract

Logistics deals with the problem of getting the right "stuff" (people, materiel, equipment, and supplies) to the right "place" at the right "time". During the last few years, it has been widely recognized that the next generation of logistics information systems must be powerful enough to manipulate massive, distributed, logistics databases, and enable logisticians to perform a variety of functions such as tracking the status of supplies and materials, planning based on the current status, efficiently tracking status changes as they occur, and re-planning as needed in order to accomplish the mission(s) at hand. In 1996, the Defense Advanced Research Projects Agency (DARPA) started an $80 million research effort called the Advanced Logistics Project (ALP) aimed at developing the next generation of logistics systems. This paper will describe the goals of ALP, the multiagent logistics architecture proposed by ALP, how this architecture supports the achievement of ALP's goals, and how it scales up to applications involving tens of thousands of cooperating software components built by multiple, disparate companies and government entities.

Keywords: Artificial intelligence, multiagent systems, planning and monitoring, logistics

1. Introduction

Logistics refers to the task of getting the "right stuff" to the "right place" at the "right time." Almost every major corporation, large organization, and complex government entity needs logistics support on a continuous, day-in, day-out basis. In addition to the planning process itself, modern logistics systems have three primary challenges:

- **Rapid Supply:** Every major company or government organization must be able to rapidly provide the assets to keep the business of that organization running smoothly. The challenge is to build a system that perfectly balances high fidelity estimation with adaptive demand routing to create a real-time hybrid supply chain which combines the "push" of supplies in

anticipation of projected demand against the "pull" of supplies based on actual demand to such precision that it virtually eliminates on-hand stocks.

- **End-to-End Movement Control:** Resources are only effective if you can get them to the place they are needed when they are needed – which makes effective and efficient transportation a critical component of both customer satisfaction and global power projection. The challenge is to build a system that supports both local and global optimization over multiple transportation modes. This system can integrate the transportation pipeline so effectively it virtually eliminates the need to stage cargo at points where it will transfer from one mode to another.. Further, this system can manage multiple concurrent large-scale operational demands, each being planned against different sets of priorities and constraints.
- **Execution Monitoring:** As the best laid plans frequently go wrong, to operate effectively in the dynamics of the real world means being able to observe, react and adapt to the real world. The challenge is to create an information system that can understand the implications of changes in the real world, understand the implications and the effects on current operations and future plans, and quickly replan in such a manner as to not invalidate the progress achieved by the actions already taken but to compensate for the changes observed through subsequent actions. Ultimately, the system needs to learn from the observations and effects to change the fundamental rules under which it operates and to adapt the character of the plans it builds.

Today, logistics is a manually intensive process that, for DOD, involves hundreds of stovepipe logistics information systems. To truly get control of the global logistics process requires harnessing the power of emerging information and automation technology.

> ***The overall goal of the DARPA Advanced Logistics Project (ALP) is to ensure the next generation of logistics systems provides an easily expandable, loosely coupled (physically), yet tightly integrated (logically) business process that meets the challenges of high fidelity logistics planning, rapid supply, end-to-end movement control and execution monitoring, autonomously and automatically to a level of detail and fidelity never before achieved. In particular, ALP will develop and demonstrate the tools and concepts which could revolutionize the logistics systems of the US Department of Defense and industry in the 21st Century.***

The ability to rapidly and effectively deploy military forces to hotspots around the world has become a critical capability for US and NATO forces. Force deployment involves the timely movement of troops, equipment, and materiel to the region of conflict. Today, it often takes 45-60 days to create a logistics plan (*Log-plan)* for a given operational requirement. ALP's goal is to develop the theory, algorithms, and implementations needed to create a high fidelity Log-plan in under an hour. To address this challenge, ALP must:

I. *Provide high degrees of visibility into the logistics pipeline and information necessary to trace resources moving through the pipeline back to the assorted logistics processes which touch that resource. Meeting this goal will ensure accuracy and thereby build confidence in the system*.

II. *Provide high fidelity, real-time status information on demand, but not require high communications bandwidth for the synchronization and transfer of information for which*

there is no demand. Meeting this goal will ensure fine-grained data management for operation in bandwidth constrained environments.

III. *Be capable of anticipatory logistics for emerging changes in the operating environment and responsive, reactive logistics for unexpected changes in the environment and operational plans. Further, the system must be adaptive in its behavior, sensitive to changing policies and priorities, and continuously evolutionary in capabilities. Meeting this goal will ensure the system does not become obsolete as we continue to grow and evolve our business processes.*

IV. *Be sensitive to the pressures, priorities, and constraints of multiple concurrent operations and a spectrum of concurrent processes. Meeting this goal will ensure the system can perform the appropriate tradeoffs to achieve both the best available local and global solution.*

V. *Be capable of performing multiple, varied analysis to support "what-if" considerations and contingency planning in an effective and scalable way while ensuring the system does not lose sight of the requirements of continuing operations. Meeting this goal will ensure the human element of creativity and design have the underlying analytic support to ensure sound decision making even under extreme situations.*

VI. *Support dynamic replanning in a scalable way, given the fact that changes in the state of the supply chain may effect hundreds, or even thousands, of plans and changes occurring hundreds of times a minute. Meeting this goal will ensure the system is designed against super-linear complexity growth and chaotic instability --- both necessary for very-large scale distributed agent societies.*

VII. *Ensure capability and visibility from all echelons, from all classes of devices, operating effectively in a globally distributed, intermittent communications environment. Meeting this goal will ensure even those users with a PDA, web-browser and a cell phone can be tied into the global logistics system of the future.*

In addition, this futuristic logistics system must have the following features:

- It must be *heterogeneous*, given the US Department of Defense depends on thousands of organizations and contractors, many of whom have their own proprietary data and software systems.
- It must support *cooperation between software modules*, as programs must be able to *interoperate* with other programs and *semantically comprehend* the messages flowing between these cooperating modules.
- It must be *massively scalable* as each corporate entity may use hundreds of data sources and code bases. Hundreds of thousands, and perhaps even millions of such modules need to cooperate with one another.

2. The ALP Cluster Architecture

Since the DOD is made up of hundreds of organizations, each with its own specific planning and logistics needs, and as these organizations in turn need to interact with contractors and vendors, each of whom often have their own logistics systems, the ALP architecture had to be distributed, flexible, adaptive, autonomous and cooperative. The ALP architecture uses an *abstraction* of a logistics system's functional behavior. The idea was that if all interacting logistics systems are

viewed as *instances* of this abstraction, then we could take advantage of the known structure and capabilities of the abstraction to facilitate interoperability and cooperation among such instances. The specific design of the system was built on existing concepts in agent architectures. However, we added another layer, a cognitive planning abstraction, to better mirror the human processes the system would be representing.

The selected abstraction was called a *cluster*, which is the general framework used by each agent in the system. Once populated with specific domain behaviors and knowledge (through the insertion of "plug-ins"), the cluster is considered instantiated as an agent. Each cluster has the architecture shown in Figure 1 below.

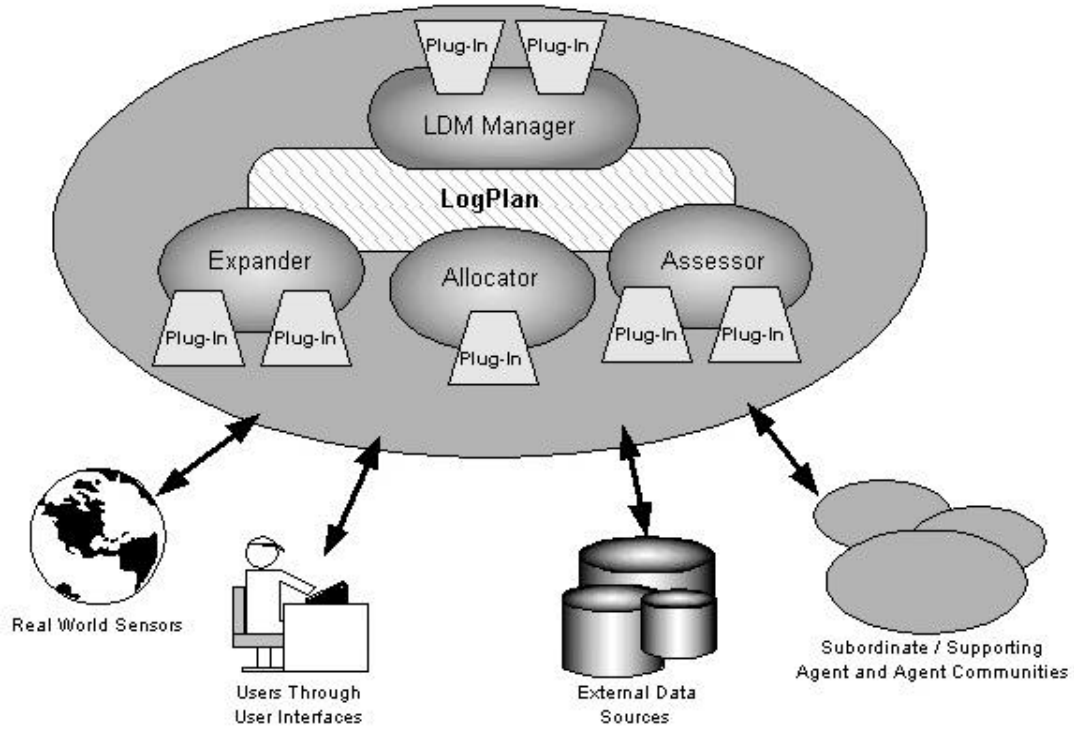

Figure 1: ALP Cluster Architecture

In the ALP architecture, every cluster represents one or more logistics business processes. The granularity and fidelity of the processes depends on a number of factors, but through encapsulation by the agent interfaces many different fidelity business processes can cooperate in the same system. For example, the Military Traffic Management Command (MTMC) may have a cluster agent that creates overland transportation plans. The Military Sealift Command (MSC) may have a cluster agent that creates and manages ships and develops ship schedules. While each agent performs its own internal planning, it also coordinates on the details where the plans intersect. For example, the coordination of truck and rail schedules from MTMC with the port arrival and ship loading schedules from MSC. In a well constructed plan, these schedules will be aligned so no time is lost aggregating or staging cargo when items being shipped are transferred from one mode of transportation (ground) to another (sea). The ALP architecture addresses both the intra-agent

planning and execution processes as well as inter-agent process coordination. This is supported by the cognitive model layer, composed of the Task Expander, Task Allocator and Task Assessor. Inter-agent coordination is supported through a well-defined task negotiation processes and will be addressed in a later section.

The Task Expander takes high level tasks and decomposes them into the specified and implied tasks that must be accomplished to achieve the higher level task. Since there are many reasonable decompositions, the Expander uses information on the world state, domain rules as well as task preferences and constraints, to find the most effective decomposition for that task. The logical, temporal and spatial arrangement of the derived tasks form a task workflow.

The Task Allocator takes the specified and implied tasks derived from the Expander decomposition, and allocates them to the available resources. These resources may be internal processes and services, subordinate and supporting agent services, or requests for user actions. The Allocator attempts to find the best allocation of resources for the components of the workflow available, given the world state and propagated task constraints and preferences.

Once the expansion and allocations are completed, the Task Assessor tracks the execution status of all tasks to ensure they are meeting their task objectives. It also continually assesses whether the workflow of tasks is meeting the original task objectives from which the workflow was derived. When the execution of a task deviates beyond a certain threshold, the Assessor component generates an *exception* that causes one or more actions to be triggered and appropriate components of the workflow to be replanned.

2.1. The Cognitive Layer

The cognitive layer, composed of the Task Expander, Task Allocator and Task Assessor, is modeled after the way humans plan. When humans plan we use "a divide and conquer approach" and decompose activities into more doable pieces. Each piece is then processed to some level and we assess the expectation that success of the individual pieces will yield success in the original higher-level task. Once convinced we have a reasonable plan, we begin execution. During execution, we continually assess each piece to see if it is achieving its individual objective and ultimately, the higher level objective.

Let's trace an example as implemented in this cognitive model through the ALP architecture. The process starts with the arrival of a task of the form of:

[Action: TRANSPORT
What: 1BN, 3rd Infantry Division
From: Ft. Stewart, USA to ReceptionPointCharlie, Saudi Arabia
By: C+30
Pref: Minimize Cost]

Generally, the most economical manner to move forces is by sea. If we assume it is currently at or before C+0, we have enough time in the schedule to use that mode. Thus the Task Expander will decompose the base "TRANSPORT" task into three sub-tasks.

TRANSPORT_GROUND From: Ft. Stewart, USA to UnNamed Port, USA

TRANSPORT_SEA From: UnNamed Port, USA to UnNamed Port, Saudi Arabia
TRANSPORT_GROUND From: UnNamed Port, Saudi Arabia to
ReceptionPointCharlie, Saudi Arabia

These tasks would be assembled as a workflow with the all the appropriate information from the parent task as well as additional logical, temporal and spatial constraints required to govern the transition between tasks. The workflow would then be posted to the Log-plan.

In our example, there are both timing and spatial constraint transitions between the various tasks. Thus, the finish location of forces (S) at the finish time (T) upon completion of the Transport_Ground task must be within an allowable 'distance' from the start location (S) and start time (T) of the Transport_Sea task. Part of the workflow constraint identifies the specific allowable time and space parameters for this task.

Next the Allocator would be alerted that there are unallocated tasks in the Log-plan. The Allocator would then attempt to match up each task Verb and necessary phrases with either an internal execution capability or a registered capability of a subordinate or supporting agent. In the example above, TRANSPORT_GROUND tasks for 'From' prepositions, which specify US locations are dispatched to the Conus Ground Agent. All TRANSPORT_SEA tasks are sent to Global Sea and TRANSPORT_GROUND tasks for 'From' prepositions outside the US are sent to the appropriate Theater Ground agent – in this case the Theater Ground agent in the CENTCOM community.

Thus the Allocator would send each of these tasks to other cluster agents for processing. Part of that processing would be the selection of ships and ports by Global Sea, which would then drive the planning of the other two ground mode agents. Though not shown above, the sub-tasks generally have all the relevant information from parent task as well any additional constraints the parent agent elects to impose.

As each of the subordinate agents respond to their individual task requests, the parent Assessor will determine how each proposed solution satisfies the plan. The parent will also coordinate additional information regarding the plan between "its children" through the refinement of task parameters such as defining the Conus port selection.

The resulting workflow, shown in Figure 2, shows the linkage between the sub-tasks in the workflow and received tasks in the child agents.

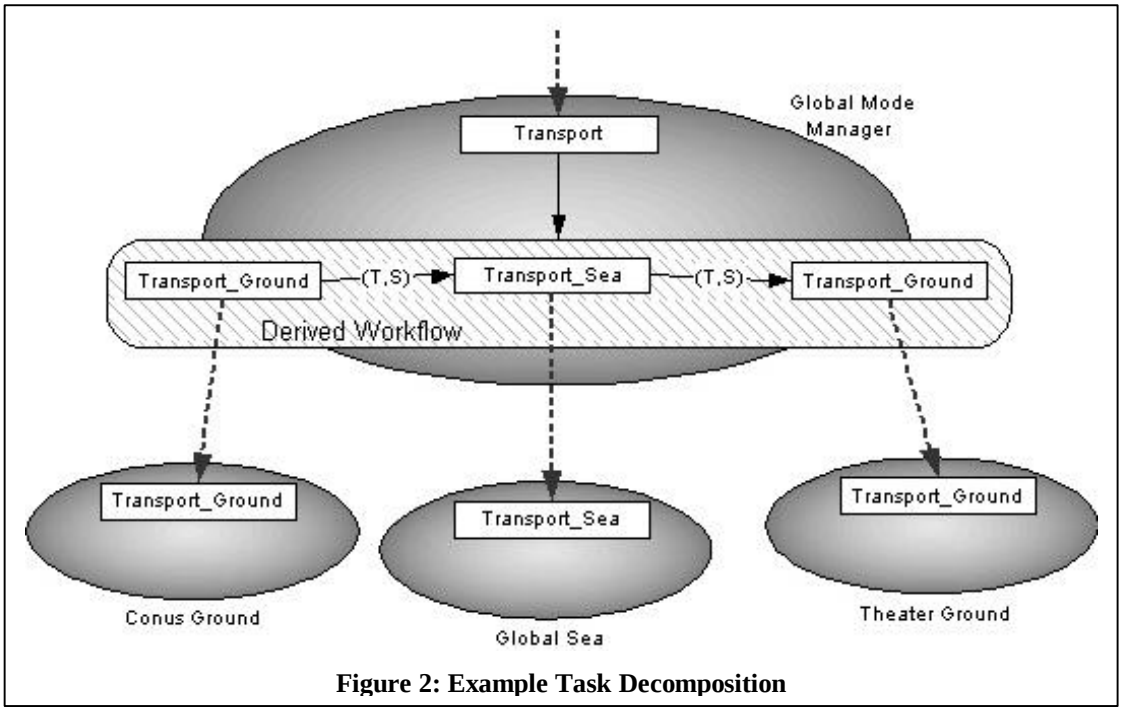

Figure 2: Example Task Decomposition

The Assessor has an additional role during execution --- that of ensuring everything meets the requirements of the plan and that the plan accomplishes the objective. The first case is concerned with environmental effects of execution --- like whether or not the ship was late due to bad weather. If the ship is late, it will miss its start date and could adversely effect the results of the plan. In the second case, the Assessor is looking for cases where the individual subtasks are successful, but they are failing in the aggregate affect. The Assessor component continually monitors all plans within the cluster, and attempts to *identify deviations* between the actual status of affairs, and the planned state of affairs. Once a deviation has been identified, the Assessor determines what to do about the deviation. In some cases, the deviation might be considered insignificant. In other cases, the deviation may trigger a rapid, *dynamic replanning* step. In yet other cases, the deviation might trigger replanning, accompanied by dynamic notifications being sent to affected parties notifying them that their shipment will be delayed.

2.2. Cluster Interaction

The ALP architecture was designed in such a way that hundreds of thousands of software agents could interoperate with one another. Each agent is a special kind of software agent [GK94], instantiated from the cluster architecture shown in Figure 1. To date, few reports exist on the deployment of massive multiagent applications. ALP's effort currently involves 270 agents, performing over 2500 business processes, in a coordinated, cooperative society of agents.

Cluster agents interoperate with each other using a task grammar-based message protocol. The task is the fundamental structure of the "request portion" of the agent negotiation dialog. The

structure of the task is a sentence in the language of discourse for ALP agents. The operational parameters of a task are contained as essential immutable contents of the task itself. In ALP, a task consists of the following data:

- ***Verb*** : A mnemonic verb name indicating the type of service requested (e.g. TRANSPORT, SUPPLY, MANAGE, SUPPORT)
- ***PrepositionalPhrases*** : A set of prepositional phrases, associating the mnemonic preposition (e.g. FROM, TO, WITH, FOR, USING, CONTAINING) with a particular object from the sending cluster's PLAN (e.g. FROM <Source Location>, TO <Destination Location>, USING <Particular Policy Guidelines>)

In addition to these strict expressions of requirements, a tasking organization can specify a series of preferences. A preference consists of the following data:

- ***Aspect*** : A dimension of measurement about which a preference is being expressed. ALP contains a predefined set of these, including START_TIME, END_TIME, COST, QUANTITY, READINESS, and CUSTOMER_SATISFACTION.
- ***ScoringFunction***: A mapping from the aspect measurement space into a dimensionless score space. This function allows for specification of best and worst points in the space of solutions in a given aspect dimension, preferred values, and, by convention, unacceptable values.
- ***Weight***: A scalar allowing for inter-preference comparison and aggregating scores from individual preferences

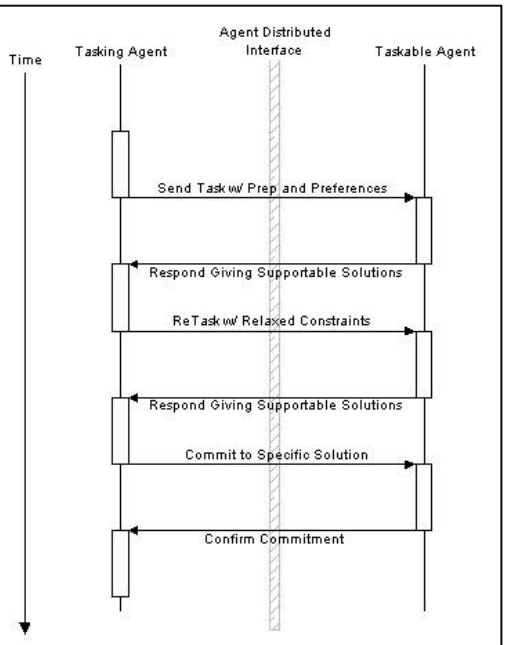

Figure 3: Use Case Diagram of Multi-Agent Task Negotiation

The communication between any two clusters takes the form of a multi-step negotiation between honest, cooperative parties. The processes starts with the transmission of the Task and its corresponding Prepositions and Preferences from the Tasking to the Taskable agent. The Taskable agent will either find the specific solution requested, or work through relaxing the task requirements in an attempt to find an acceptable solution. Relaxation begins with the preferences of the task, lowest priority preference first. Should total relaxation of the preferences fail to yield a solution, then the constraints will start to be relaxed. The Taskable agent reports back to the Tasking agent identifying the space of solutions it can support. The Tasking agent will either commit to a point in that solution space or reformulate the Task based on the penalty function information returned from the Taskable agent. The penalty function provides penalty cost values against each of the constraints

and preferences to convey which aspects of the task are proving to be untenable. With this information, the Tasking agent can participate in the refinement processes without needing access to the Taskable agent's internal state information. This interchange continues until the two agents cannot refine the task – no solution is possible – or a solution is found and committed to by both parties.

2.3. Data Management

As discussed previously, the Log-plan is the information foundation of the ALP architecture. The Log-plan is constructed from a prototype-based Logical Data Model (LDM) and fed updated information continuously by a variety of LDM Plug-ins that interfaces with sensors, databases and users in the real world (as shown in Figure 1). This section will discuss the ALP approach to information management and provide a brief overview of the prototype model approach.

Solving the complex queries from a wide range of sources requires the use of a *mediator* ([WI93] Dogac *et.al.* [DDO98]) that interoperates between multiple data sources. The mediation tool must be able to take a user query as input, and convert it into a set of sub-queries. Each of these sub-queries must have a structure that can access the data sources involved. To solve this problem, the ALP architecture has the LDM Manager act as a mediator over the set of information sources that are defined in the available plug-in set.. Each plug-in will represent an interface to one or more information sources. Each plug-in registers the type and context of queries it can perform. As other components, like the Task Expander, access elements of the Log-plan, these accesses form a special class of query. If the information is present and current, the values in the Log-plan are returned. If they are missing or expired, the LDM Manager is triggered to obtain or refresh the data. Many of the plug-ins are simply SQL or JDBC interfaces into databases, with some additional information to map them to or from their native structures to the Log-plan objects.

The second critical component to effective data management is the LDM, which in effect, forms the class model for the Log-plan. The current ALP implementation includes 270 interacting agents executing over 2500 business processes --- one of the biggest multiagent applications built to date. The Log-plan for this society of agents is *enormous* – logically representing over 4million distinct object-types for supplies and equipment alone, and well over 6 million distinct object-types altogether. But the underlying pseudo-class model, defined by the LDM, has less than 500 classes including all the unique role-based behavior components. To achieve this level of flexibility and efficiency, the ALP architecture has used a specialization of the prototype (delegation) design pattern in the construction of the underlying model.

The approach taken to represent this huge set of types includes a mix of hierarchical decomposition (i.e. inheritance) and prototyping methods. In prototyping (Lieberman [L86] "a prototype represents the default behavior for a concept, and new objects can re-use part of the knowledge stored in the prototype by saying how the new object differs from the prototype" [L86]. Prototypes are used in the LDM to store small variations between types. For example, prototypes are used in ALP to store the fact that both C5's and C130's are fixed-wing aircraft, but they are slightly different in some respects.

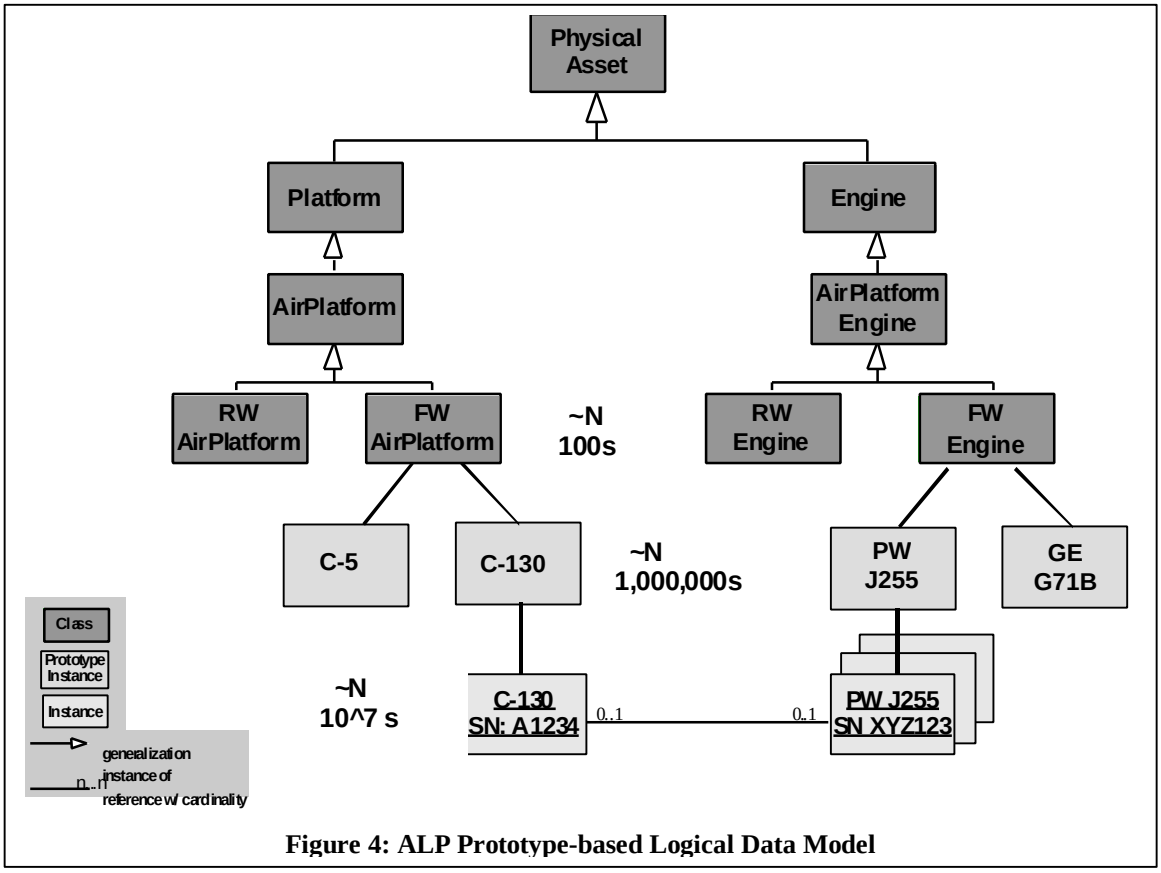

Figure 4: ALP Prototype-based Logical Data Model

The ALP LDM embodies a *mix* of prototyping and inheritance. Figure 3 above shows the representation of types related to fixed-wing aircraft, using a mix of prototypes and inheritance. In this figure, we notice that C5 and the C130 are both *prototypes* of the "FW AirPlatform" class. At the abstract "FWAircraft" class level, we have only hundreds of object classes yet can represent millions of object prototypes. Similarly at the next level of this hierarchy, instead of having millions of unique objects, we have millions of distinct *instances each of which draws its behavior from its prototypical instance*. Thus class instances are constructed from a small number of base classes and a combination of capability and role decorations, packaged as expandable prototypes, which can be specialized at run time as needed. This simple but powerful approach not only allows a much reduced object model but also allows the dynamic adaptation of capabilities at run time. Thus, as a truck moves from land onto a ship, its role and corresponding behaviors change from a conveyance to cargo.

3. Comparing the ALP Architecture

Having presented the basic design approach and characteristics of the ALP architecture, it is worthwhile to compare and contrast this design to other architectural approaches. Though this is a brief comparison, it is quickly apparent that the ALP architecture attempts to pull together many of the best features and concepts of some of the most popular and effective architectures, while adding additional features that make it a truly unique architectural design. The resulting synergy of design has provided a powerful agent architecture which, to date, has been highly scalable and extremely flexible. This section identifies the similarities and differences between the ALP

architecture and conventional planning architectures, reactive architectures like Brook's subsumption architecture [BR86], component-based architectures, and finally the emerging concepts in BDI architecture.

3.1 Planning Architectures

The cognitive layer discussed earlier in this paper is based on concepts of hierarchical planning that have been used in AI planning since the early STRIPS [FN71] work in the 1970's. The major differences are that the planning domains are encapsulated within a plug-in and that a plug-in plans over just the task space for which it is registered --- much like a distributed planning system. Further, the planning phase, performed in the Expander, is segregated from the resource allocation phase, performed by the Allocator. This forms a logical breadth-first planning approach with delayed resource commitment. Further, both the task decomposition and resource allocation is subject to further refinement based on commitments in other parts of the plan, similar to the iterative refinement planning approach. The workflow structure has many structural similarities to a partial order, analysis based correction infrastructure similar to that of NONLIN [TA77].

One of the real powers in ALP is the planning infrastructure in the architecture that serves primarily as a router and controller, allowing the planning of individual plug-ins to employ any style of planning or scheduling that proves most effective for that problem. In the current implementation, we have a variety of simple STRIPS-like planners, expert system planners, case-based planners, genetic algorithm schedulers, and numerous others. In some cases, the plug-in is most simplified using a particular planning approach. In other cases, the problem domain makes a particular approach significantly more efficient. By providing a hierarchical planning infrastructure but having total flexibility on the planning at the branch and leaf nodes, we can build an effective heterogeneous planning system.

3.2 Subsumption Architectures

Since we are trying to build a system that will support execution, there are a number experiences that we have drawn upon from the robotics community on how to do execution effectively. Some of the most effective execution architectures are those based on the subsumption approach proposed by Brooks [BR86]. In this approach, multiple concurrent plans are executed, usually using some variant of a state machine, arranged by priority. Higher priority plans intercede under appropriate conditions to short circuit the execution and invoke an immediate response, as in the case of a potential collision situation. Lower priority plans are the ones focused on the mission objectives of the robot - like finding a soda can. The power in this approach is the ability to react during execution without having to do any mapping of the situation into complex representations or performing any advanced reasoning. These layers operate as independent services addressing a particular business process.

The Assessor component of the architecture, though not originally designed with the subsumption concept in mind, turns out be a very effective subsumption implementation. The Assessors look for patterns in the information of the Log-plan and the information flowing into the agent from external sources. Each Assessor acts like its own state machine addressing particular forms of failure. Based on the type of failure mode and priority of the last, the effective priority of an assessed failure forms a layered reactive response system. Since some high priority failure modes have immediate actions associated with them, the result is a reactive system for critical failures and

controlled replanning cycle for non-critical failures. The logic for the determination of critical and non-critical, as well as the patterns for assessment, can be state dependent and adaptive.

Unlike a strict subsumption approach, some of the Assessor plug-ins utilize the logistics plan, which is in effect a model of the shared world-state as reconstructed by the agent society. Though not all Assessors use the Log-plan, it has proven timely and accurate for all but the most time-sensitive failure information. Like the subsumption approach, the plug-in does not maintain any state outside that captured in the information of the Log-plan. This, in effect, supports the classification of the ALP architecture as a hybrid deliberate-reactive system. As we move more into the real-time execution domain, it is useful to characterize the requirements of the assessment function as either deliberate or reactive, and construct the Assessor accordingly.

3.3 Component Architectures

The ALP architecture has strongly embraced the component-based design, an approach that has been getting so much attention, recently. The design of both the internal architecture service components and the plug-in interfaces is very similar to the current Java Beans [VO98][DS99] approach to behavior encapsulation. While ALP has a much stronger semantics translation component, the idea of communicating services, registering, and brokering is very similar to the Jini [AO99] services approach.

The concept of the plug-in is to allow a standard mechanism, constructed as an API, to connect specific domain behaviors. These specific behaviors could be in the form of an algorithm, an application, or an entire system. From the rest of the system, there is no differentiation between the magnitude of the code at the other side of the plug-in. Further, the design of the plug in interface code provides an easy mechanism to evolve a heterogeneous system over time without disrupting operations, an extension of the Java class loader concept.

3.4 BDI Architectures

Another popular architecture for agent development is the Belief-Decision-Intent architectures [KG91][RG92][WE99]. These architectures pursue practical machine reasoning by explicit differentiation of the concepts of beliefs, decisions and intent in the planning and deliberation process. There are a number of similarities between the ALP approach and BDI architectures and a few key differences.

The BDI architecture is careful to make a distinction between what goals are being pursued (intent) and how these goals are going to be pursued (decision). There is an understanding that executing the decisions will not necessarily allow you to achieve the intent. The ALP architecture has a similar separation in that the root task comes into the agent, or is generated by the agent, in the form of a task objective, similar to an intent. From that the Expanders will generate a workflow which will define the 'how', or decision plan. Some of those same tasks will be passed to other agents which then interpret them as their objectives. In a slightly different manner, the Expander is defining the 'what' in the ALP approach and the Allocator plug-in is responsible for determining the 'how' by virtue of the assets allocated, schedules generated, or subordinates assigned. Though in this instance, the 'how' component is broken into two pieces, the specification of the activity by the Expander and the association of that activity with the resources that can accomplish it through the Allocator.

In the BDI architecture, beliefs are generated from reasoning over the world model. Beliefs represent the set of acceptable plans that an agent believes will allow it to achieve its intentions. From that set, one plan is selected as the best according to some criteria - and that becomes the decision. For planning within an ALP agent, that process is localized to an Expander which is free to employ any mechanism it desires to develop a single plan which will be posted as part of the Log-Plan. If the planning process involves more than one agent, then an approach very similar to the Beliefs process of BDI is used. In the ALP system, the task objective is passed to the subordinate agent, which then responds with a set of penalty functions representing the solution space, usually consisting of multiple ways to meet the objective. Note, unlike BDI, what is exchanged is the penalty function relating the relative costs along the preference and criteria vectors for various solutions. No plan solution or state information is shared between the parent and child. Based on the penalty function information only, the parent selects a desired point on the graph and thereby decides upon a given plan.

4. Future Plans

In the remaining two years of the Advanced Logistics Project, the architecture will be expanded to support the development of multiple concurrent plans and the creation of multiple worlds. These extensions will allow an agent or set of agents to develop contingency plans, evaluate alternate courses of action, address uncertainty in operational outcomes, and change the fundamental operating assumptions upon which the logistics plans are constructed. With this capability, the system will be better equipped to perform 'what-if' analysis of various forms and maintain concurrent plan alternatives.

5. Conclusions

In this paper, we have briefly described the overall architecture of the DARPA Advanced Logistics Project. The main goal of ALP is to develop and demonstrate the architecture, theory, and algorithms needed for the next generation global logistics systems. In the DOD domain, this involves the ability to completely automate the creation of a logistics plan and reduce plan development time from the current 45-60 days to less than an hour.

The accomplishment of this goal requires the ability to solve a variety of hard computational problems, ranging from the access of multiple, distributed, heterogeneous databases, to the development of scalable plan sentinels and agent infrastructures that can support hundreds of thousands of interacting agents. This paper has presented the ALP architecture attempted to show how it supports the goals of the program, and how it compares to several other agent architecture approaches.

REFERENCES

[AO99] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. The Jini Specification, Addison-Wesley, 1999.

[BR86] R.A. Brooks. *A Robust Layered Control System For A Mobile Robot*, IEEE Journal of Robotics and Automaton, 2(1):14-23, 1986.

[DRS98] A. Dekhtyar, R. Ross and V.S. Subrahmanian. Probabilistic Temporal Databases: Algebra and Implementation, Univ. of Maryland Technical Report CS-TR-3987, Jan.1999.

[DDO98] A. Dogac, C. Dengi and M.T. Özsu, *Building Interoperable Databases on Distributed Object Management Platforms*, Communication of the ACM, 41(9): 95-103, September 1998.

[DS98] C. E. Dyreson and R. T. Snodgrass. *Supporting Valid-time Indeterminacy*, ACM Transactions on Database Systems, March 1998.

[DS99] T. Daly, U. Shetty. *Enterprise JavaBeans Tutorial: Building Your First Stateless Session Bean,* Java Tutorials, Sun, 1999.

[ESP98] T. Eiter, V.S. Subrahmanian and G. Pick. *Heterogeneous Active Agents, I: Semantics*, Artificial Intelligence Journal, Vol. 108, Nr. 1, pps 179-255.

[FN71] R.E. Fikes, N.J. Nilsson, *STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving*, Artificial Intelligence, 2, pp. 189-208.

[GK94] M.R. Genesereth and S.P. Ketchpel. *Software Agents*. Communications of the ACM, 37(7), 1994.

[KG91] D. Kinny and M. Georgeff. *Commitment and Effectiveness of Situated Agents*, Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), pages 82-88, 1991.

[L86] H. Lieberman. *Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems.* OOPSLA '86 Proceedings. ACM Press, September 1986.

[NI80] N.J. Nilsson. Principles of Artificial Intelligence, Morgan Kaufmann, 1980.

[RG92] A.S. Rao and M.P. Georgeff. *An Abstract Architecture for Rational Agents*, Proceedings of Knowledge Representation and Reasoning (KR&R-92), pages 439-449, 1992.

[SRM98] J.H. Schafer, T.J. Rogers and J. Marin. (1998) *Networked Visualization of US Army War Reserve Readiness Data*, in Proc. 1998 Intl. Workshop on Multimedia Information Systems (eds. S. Jajodia, M.T. Ozsu and A. Dogac), Springer Lecture Notes in Computer Science Vol., 1508, pps 136-147.

[SU99] V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan and R. Ross. Heterogenous Agent Systems: Theory and Implementation, MIT Press, Jan. 2000, to appear.

[TA77] A. Tate. *Generating Project Networks*, IJCAI-77, Boston, MA.

[VO98] G. Voss. *Introducing Java Beans*, Sun, 1998.

[WI93] G. Wiederhold. *Intelligent Integration of Information*, In: Proc. 1993 ACM SIGMOD Conf. on Management of Data, pp 434--437, 1993.

[WE99] G. Weiss. Multi-Agent Systems: A Modern Approach To Distributed Artificial Intelligence, MIT Press, 1999.